
R Package *hdnom*
Function Reference Manual

Nan Xiao, Qing-Song Xu, Miao-Zhu Li

hdnom version 4.0

July 25, 2016



hdnom.org

Documented R Functions

hdnom-package	3
hdcox.aenet	3
hdcox.lasso	4
hdcox.enet	5
hdcox.flasso	7
hdcox.lasso	8
hdcox.mcp	9
hdcox.mnet	10
hdcox.scad	11
hdcox.snet	12
hdnom.calibrate	14
hdnom.compare.calibrate	16
hdnom.compare.validate	17
hdnom.external.calibrate	18
hdnom.external.validate	20
hdnom.kmplot	21
hdnom.logrank	22
hdnom.nomogram	24
hdnom.validate	25
hdnom.varinfo	27
plot.hdnom.calibrate	28
plot.hdnom.compare.calibrate	29
plot.hdnom.compare.validate	29
plot.hdnom.external.calibrate	30
plot.hdnom.external.validate	31
plot.hdnom.nomogram	31
plot.hdnom.validate	32
predict.hdcox.model	32
print.hdcox.model	33
print.hdnom.calibrate	34
print.hdnom.compare.calibrate	34
print.hdnom.compare.validate	35
print.hdnom.external.calibrate	35
print.hdnom.external.validate	36
print.hdnom.nomogram	36
print.hdnom.validate	37
smart	37
smarto	39
summary.hdnom.calibrate	40
summary.hdnom.compare.calibrate	41
summary.hdnom.compare.validate	41
summary.hdnom.external.calibrate	42
summary.hdnom.external.validate	42
summary.hdnom.validate	43

hdnom-package	<i>Nomograms for High-Dimensional Data with Penalized Cox Models</i>
---------------	--

Description

Nomograms for High-Dimensional Data with Penalized Cox Models

Details

The vignette can be opened with `vignette('hdnom')`.

hdcox.aenet	<i>Adaptive Elastic-Net Model Selection for High-Dimensional Cox Models</i>
-------------	---

Description

Automatic adaptive elastic-net model selection for high-dimensional Cox models, evaluated by penalized partial-likelihood.

Usage

```
hdcox.aenet(x, y, nfolds = 5L, alphas = seq(0.05, 0.95, 0.05),
  rule = c("lambda.min", "lambda.1se"), seed = c(1001, 1002),
  parallel = FALSE)
```

Arguments

<code>x</code>	Data matrix.
<code>y</code>	Response matrix made with Surv .
<code>nfolds</code>	Fold numbers of cross-validation.
<code>alphas</code>	Alphas to tune in cv.glmnet .
<code>rule</code>	Model selection criterion, "lambda.min" or "lambda.1se". See cv.glmnet for details.
<code>seed</code>	Two random seeds for cross-validation fold division in two estimation steps.
<code>parallel</code>	Logical. Enable parallel parameter tuning or not, default is FALSE. To enable parallel tuning, load the <code>doParallel</code> package and run <code>registerDoParallel()</code> with the number of CPU cores before calling this function.

Examples

```

library("survival")
library("rms")

# Load imputed SMART data
data("smart")
x = as.matrix(smart[, -c(1, 2)])
time = smart$TEVENT
event = smart$EVENT
y = Surv(time, event)

# To enable parallel parameter tuning, first run:
# library("doParallel")
# registerDoParallel(detectCores())
# then set hdcx.aenet(..., parallel = TRUE).

# Fit Cox model by adaptive elastic-net penalization
aenetfit = hdcx.aenet(x, y, nfolds = 3, alphas = c(0.3, 0.7),
                    rule = "lambda.1se", seed = c(5, 7))

# Prepare data for hdnom.nomogram
x.df = as.data.frame(x)
dd = datadist(x.df)
options(datadist = "dd")

# Generate hdnom.nomogram objects and plot nomogram
nom = hdnom.nomogram(aenetfit$aenet_model, model.type = "aenet",
                    x, time, event, x.df,
                    lambda = aenetfit$aenet_best_lambda, pred.at = 365 * 2,
                    funlabel = "2-Year Overall Survival Probability")

plot(nom)

```

hdcx.lasso	<i>Adaptive Lasso Model Selection for High-Dimensional Cox Models</i>
------------	---

Description

Automatic adaptive lasso model selection for high-dimensional Cox models, evaluated by penalized partial-likelihood.

Usage

```

hdcx.lasso(x, y, nfolds = 5L, rule = c("lambda.min", "lambda.1se"),
          seed = c(1001, 1002))

```

Arguments

<code>x</code>	Data matrix.
<code>y</code>	Response matrix made by Surv .
<code>nfolds</code>	Fold numbers of cross-validation.
<code>rule</code>	Model selection criterion, "lambda.min" or "lambda.1se". See cv.glmnet for details.
<code>seed</code>	Two random seeds for cross-validation fold division in two estimation steps.

Examples

```

library("survival")
library("rms")

# Load imputed SMART data
data("smart")
x = as.matrix(smart[, -c(1, 2)])
time = smart$TEVENT
event = smart$EVENT
y = Surv(time, event)

# Fit Cox model by adaptive lasso penalization
alassofit = hdcox.lasso(x, y, nfolds = 3, rule = "lambda.1se", seed = c(7, 11))

# Prepare data for hdnom.nomogram
x.df = as.data.frame(x)
dd = datadist(x.df)
options(datadist = "dd")

# Generate hdnom.nomogram objects and plot nomogram
nom = hdnom.nomogram(alassofit$alasso_model, model.type = "alasso",
                    x, time, event, x.df,
                    lambda = alassofit$alasso_best_lambda, pred.at = 365 * 2,
                    funlabel = "2-Year Overall Survival Probability")

plot(nom)

```

hdcox.enet

*Elastic-Net Model Selection for High-Dimensional Cox Models***Description**

Automatic elastic-net model selection for high-dimensional Cox models, evaluated by penalized partial-likelihood.

Usage

```

hdcox.enet(x, y, nfolds = 5L, alphas = seq(0.05, 0.95, 0.05),
          rule = c("lambda.min", "lambda.1se"), seed = 1001, parallel = FALSE)

```

Arguments

<code>x</code>	Data matrix.
<code>y</code>	Response matrix made by Surv .
<code>nfolds</code>	Fold numbers of cross-validation.
<code>alphas</code>	Alphas to tune in cv.glmnet .
<code>rule</code>	Model selection criterion, "lambda.min" or "lambda.1se". See cv.glmnet for details.
<code>seed</code>	A random seed for cross-validation fold division.
<code>parallel</code>	Logical. Enable parallel parameter tuning or not, default is FALSE. To enable parallel tuning, load the <code>doParallel</code> package and run <code>registerDoParallel()</code> with the number of CPU cores before calling this function.

Examples

```

library("survival")
library("rms")

# Load imputed SMART data
data("smart")
x = as.matrix(smart[, -c(1, 2)])
time = smart$TEVENT
event = smart$EVENT
y = Surv(time, event)

# To enable parallel parameter tuning, first run:
# library("doParallel")
# registerDoParallel(detectCores())
# then set hdcox.enet(..., parallel = TRUE).

# Fit Cox model by elastic-net penalization
enetfit = hdcox.enet(x, y, nfolds = 3, alphas = c(0.3, 0.7),
                    rule = "lambda.1se", seed = 11)

# Prepare data for hdnom.nomogram
x.df = as.data.frame(x)
dd = datadist(x.df)
options(datadist = "dd")

# Generate hdnom.nomogram objects and plot nomogram
nom = hdnom.nomogram(enetfit$enet_model, model.type = "enet",
                    x, time, event, x.df,
                    lambda = enetfit$enet_best_lambda, pred.at = 365 * 2,
                    funlabel = "2-Year Overall Survival Probability")

plot(nom)

```

`hdcx.flasso`*Fused Lasso Model Selection for High-Dimensional Cox Models*

Description

Automatic fused lasso model selection for high-dimensional Cox models, evaluated by penalized partial-likelihood.

Usage

```
hdcx.flasso(x, y, nfolds = 5L, seed = 1001, trace = FALSE)
```

Arguments

<code>x</code>	Data matrix.
<code>y</code>	Response matrix made by Surv .
<code>nfolds</code>	Fold numbers of cross-validation.
<code>seed</code>	A random seed for cross-validation fold division.
<code>trace</code>	Output the cross-validation parameter tuning progress or not. Default is <code>FALSE</code> .

Note

The cross-validation procedure used in this function is the so-called approximated cross-validation provided by the `penalized` package. Be careful dealing with the results since they might be more optimistic than fully fitting the model. This cross-validation method is more suitable for datasets with larger number of observations, and a higher number of cross-validation folds.

Examples

```
library("survival")
library("rms")

# Load imputed SMART data; only use the first 120 samples
data("smart")
x = as.matrix(smart[, -c(1, 2)])[1:140, ]
time = smart$TEVENT[1:140]
event = smart$EVENT[1:140]
y = Surv(time, event)

# Fit Cox model by fused lasso penalization
flassofit = hdcx.flasso(x, y, nfolds = 3, seed = 11)

# Prepare data for hdnom.nomogram
x.df = as.data.frame(x)
dd = datadist(x.df)
options(datadist = "dd")
```

```
# Generate hdnom.nomogram objects and plot nomogram
nom = hdnom.nomogram(flassofit$flasso_model, model.type = "flasso",
                    x, time, event, x.df,
                    lambda = flassofit$flasso_best_lambda, pred.at = 365 * 2,
                    funlabel = "2-Year Overall Survival Probability")

plot(nom)
```

hdcox.lasso

Lasso Model Selection for High-Dimensional Cox Models

Description

Automatic lasso model selection for high-dimensional Cox models, evaluated by penalized partial-likelihood.

Usage

```
hdcox.lasso(x, y, nfolds = 5L, rule = c("lambda.min", "lambda.1se"),
            seed = 1001)
```

Arguments

<code>x</code>	Data matrix.
<code>y</code>	Response matrix made by Surv .
<code>nfolds</code>	Fold numbers of cross-validation.
<code>rule</code>	Model selection criterion, "lambda.min" or "lambda.1se". See cv.glmnet for details.
<code>seed</code>	A random seed for cross-validation fold division.

Examples

```
library("survival")
library("rms")

# Load imputed SMART data
data("smart")
x = as.matrix(smart[, -c(1, 2)])
time = smart$TEVENT
event = smart$Eevent
y = Surv(time, event)

# Fit Cox model by lasso penalization
lassofit = hdcox.lasso(x, y, nfolds = 5, rule = "lambda.1se", seed = 11)

# Prepare data for hdnom.nomogram
x.df = as.data.frame(x)
```



```

dd = datadist(x.df)
options(datadist = "dd")

# Generate hdnom.nomogram objects and plot nomogram
nom = hdnom.nomogram(lassofit$lasso_model, model.type = "lasso",
                    x, time, event, x.df,
                    lambda = lassofit$lasso_best_lambda, pred.at = 365 * 2,
                    funlabel = "2-Year Overall Survival Probability")

plot(nom)

```

hdcox.mcp

MCP Model Selection for High-Dimensional Cox Models

Description

Automatic MCP model selection for high-dimensional Cox models, evaluated by penalized partial-likelihood.

Usage

```

hdcox.mcp(x, y, nfolds = 5L, gammas = c(1.01, 1.7, 3, 100), seed = 1001,
          trace = FALSE, parallel = FALSE)

```

Arguments

<code>x</code>	Data matrix.
<code>y</code>	Response matrix made by Surv .
<code>nfolds</code>	Fold numbers of cross-validation.
<code>gammas</code>	Gammas to tune in cv.ncvsurv .
<code>seed</code>	A random seed for cross-validation fold division.
<code>trace</code>	Output the cross-validation parameter tuning progress or not. Default is <code>FALSE</code> .
<code>parallel</code>	Logical. Enable parallel parameter tuning or not, default is <code>FALSE</code> . To enable parallel tuning, load the <code>doParallel</code> package and run <code>registerDoParallel()</code> with the number of CPU cores before calling this function.

Examples

```

library("survival")
library("rms")

# Load imputed SMART data; only use the first 150 samples
data("smart")
x = as.matrix(smart[, -c(1, 2)])[1:150, ]
time = smart$TEVENT[1:150]
event = smart$EVENT[1:150]

```

```

y = Surv(time, event)

# Fit Cox model by MCP penalization
mcpfit = hdcox.mcp(x, y, nfolds = 3, gammas = c(2.1, 3), seed = 1001)

# Prepare data for hdnom.nomogram
x.df = as.data.frame(x)
dd = datadist(x.df)
options(datadist = "dd")

# Generate hdnom.nomogram objects and plot nomogram
nom = hdnom.nomogram(mcpfit$mcp_model, model.type = "mcp", x, time, event, x.df,
                    lambda = mcpfit$mcp_best_lambda, pred.at = 365 * 2,
                    funlabel = "2-Year Overall Survival Probability")

plot(nom)

```

hdcox.mnet

Mnet Model Selection for High-Dimensional Cox Models

Description

Automatic Mnet model selection for high-dimensional Cox models, evaluated by penalized partial-likelihood.

Usage

```

hdcox.mnet(x, y, nfolds = 5L, gammas = c(1.01, 1.7, 3, 100),
          alphas = seq(0.05, 0.95, 0.05), seed = 1001, trace = FALSE,
          parallel = FALSE)

```

Arguments

<code>x</code>	Data matrix.
<code>y</code>	Response matrix made by Surv .
<code>nfolds</code>	Fold numbers of cross-validation.
<code>gammas</code>	Gammas to tune in cv.ncvsurv .
<code>alphas</code>	Alphas to tune in cv.ncvsurv .
<code>seed</code>	A random seed for cross-validation fold division.
<code>trace</code>	Output the cross-validation parameter tuning progress or not. Default is <code>FALSE</code> .
<code>parallel</code>	Logical. Enable parallel parameter tuning or not, default is <code>FALSE</code> . To enable parallel tuning, load the <code>doParallel</code> package and run <code>registerDoParallel()</code> with the number of CPU cores before calling this function.

Examples

```

library("survival")
library("rms")

# Load imputed SMART data; only use the first 120 samples
data("smart")
x = as.matrix(smart[, -c(1, 2)])[1:120, ]
time = smart$TEVENT[1:120]
event = smart$EVENT[1:120]
y = Surv(time, event)

# Fit Cox model by Mnet penalization
mnetfit = hdcox.mnet(x, y, nfolds = 3, gammas = 3,
                    alphas = c(0.3, 0.8), seed = 1010)

# Prepare data for hdnom.nomogram
x.df = as.data.frame(x)
dd = datadist(x.df)
options(datadist = "dd")

# Generate hdnom.nomogram objects and plot nomogram
nom = hdnom.nomogram(mnetfit$mnet_model, model.type = "mnet",
                    x, time, event, x.df,
                    lambda = mnetfit$mnet_best_lambda,
                    pred.at = 365 * 2,
                    funlabel = "2-Year Overall Survival Probability")

plot(nom)

```

hdcox.scad

*SCAD Model Selection for High-Dimensional Cox Models***Description**

Automatic SCAD model selection for high-dimensional Cox models, evaluated by penalized partial-likelihood.

Usage

```

hdcox.scad(x, y, nfolds = 5L, gammas = c(2.01, 2.3, 3.7, 200),
           seed = 1001, trace = FALSE, parallel = FALSE)

```

Arguments

<code>x</code>	Data matrix.
<code>y</code>	Response matrix made by <code>Surv</code> .
<code>nfolds</code>	Fold numbers of cross-validation.
<code>gammas</code>	Gammas to tune in <code>cv.ncvsurv</code> .

<code>seed</code>	A random seed for cross-validation fold division.
<code>trace</code>	Output the cross-validation parameter tuning progress or not. Default is FALSE.
<code>parallel</code>	Logical. Enable parallel parameter tuning or not, default is FALSE. To enable parallel tuning, load the <code>doParallel</code> package and run <code>registerDoParallel()</code> with the number of CPU cores before calling this function.

Examples

```
library("survival")
library("rms")

# Load imputed SMART data; only use the first 120 samples
data("smart")
x = as.matrix(smart[, -c(1, 2)])[1:120, ]
time = smart$TEVENT[1:120]
event = smart$EVENT[1:120]
y = Surv(time, event)

# Fit Cox model by SCAD penalization
scadfit = hdcox.scad(x, y, nfolds = 3, gammas = c(3.7, 5), seed = 1010)

# Prepare data for hdnom.nomogram
x.df = as.data.frame(x)
dd = datadist(x.df)
options(datadist = "dd")

# Generate hdnom.nomogram objects and plot nomogram
nom = hdnom.nomogram(scadfit$scad_model, model.type = "scad", x, time, event, x.df,
                    lambda = scadfit$scad_best_lambda, pred.at = 365 * 2,
                    funlabel = "2-Year Overall Survival Probability")

plot(nom)
```

hdcox.snet

Snet Model Selection for High-Dimensional Cox Models

Description

Automatic Snet model selection for high-dimensional Cox models, evaluated by penalized partial-likelihood.

Usage

```
hdcox.snet(x, y, nfolds = 5L, gammas = c(2.01, 2.3, 3.7, 200),
          alphas = seq(0.05, 0.95, 0.05), seed = 1001, trace = FALSE,
          parallel = FALSE)
```

Arguments

<code>x</code>	Data matrix.
<code>y</code>	Response matrix made by <code>Surv</code> .
<code>nfolds</code>	Fold numbers of cross-validation.
<code>gammas</code>	Gammas to tune in <code>cv.ncvsurv</code> .
<code>alphas</code>	Alphas to tune in <code>cv.ncvsurv</code> .
<code>seed</code>	A random seed for cross-validation fold division.
<code>trace</code>	Output the cross-validation parameter tuning progress or not. Default is FALSE.
<code>parallel</code>	Logical. Enable parallel parameter tuning or not, default is FALSE. To enable parallel tuning, load the <code>doParallel</code> package and run <code>registerDoParallel()</code> with the number of CPU cores before calling this function.

Examples

```

library("survival")
library("rms")

# Load imputed SMART data; only use the first 120 samples
data("smart")
x = as.matrix(smart[, -c(1, 2)])[1:120, ]
time = smart$TEVENT[1:120]
event = smart$EVENT[1:120]
y = Surv(time, event)

# Fit Cox model by Snet penalization
snetfit = hdcx.snet(x, y, nfolds = 3, gammas = 3.7,
                  alphas = c(0.3, 0.8), seed = 1010)

# Prepare data for hdnom.nomogram
x.df = as.data.frame(x)
dd = datadist(x.df)
options(datadist = "dd")

# Generate hdnom.nomogram objects and plot nomogram
nom = hdnom.nomogram(snetfit$snet_model, model.type = "snet",
                    x, time, event, x.df,
                    lambda = snetfit$snet_best_lambda,
                    pred.at = 365 * 2,
                    funlabel = "2-Year Overall Survival Probability")

plot(nom)

```

hdnom.calibrate

Calibrate High-Dimensional Cox Models

Description

Calibrate High-Dimensional Cox Models

Usage

```
hdnom.calibrate(x, time, event, model.type = c("lasso", "alasso", "flasso",
  "enet", "aenet", "mcp", "mnet", "scad", "snet"), alpha, lambda,
  pen.factor = NULL, gamma, method = c("fitting", "bootstrap", "cv",
  "repeated.cv"), boot.times = NULL, nfolds = NULL, rep.times = NULL,
  pred.at, ngroup = 5, seed = 1001, trace = TRUE)
```

Arguments

<code>x</code>	Matrix of training data used for fitting the model; on which to run the calibration.
<code>time</code>	Survival time. Must be of the same length with the number of rows as <code>x</code> .
<code>event</code>	Status indicator, normally 0 = alive, 1 = dead. Must be of the same length with the number of rows as <code>x</code> .
<code>model.type</code>	Model type to calibrate. Could be one of "lasso", "alasso", "flasso", "enet", "aenet", "mcp", "mnet", "scad", or "snet".
<code>alpha</code>	Value of the elastic-net mixing parameter alpha for <code>enet</code> , <code>aenet</code> , <code>mnet</code> , and <code>snet</code> models. For <code>lasso</code> , <code>alasso</code> , <code>mcp</code> , and <code>scad</code> models, please set <code>alpha = 1</code> . <code>alpha=1</code> : lasso (l1) penalty; <code>alpha=0</code> : ridge (l2) penalty. Note that for <code>mnet</code> and <code>snet</code> models, <code>alpha</code> can be set to very close to 0 but not 0 exactly.
<code>lambda</code>	Value of the penalty parameter lambda to use in the model fits on the resampled data. From the Cox model you have built.
<code>pen.factor</code>	Penalty factors to apply to each coefficient. From the built <i>adaptive lasso</i> or <i>adaptive elastic-net</i> model.
<code>gamma</code>	Value of the model parameter gamma for MCP/SCAD/Mnet/Snet models.
<code>method</code>	Calibration method. Options including "fitting", "bootstrap", "cv", and "repeated.cv".
<code>boot.times</code>	Number of repetitions for bootstrap.
<code>nfolds</code>	Number of folds for cross-validation and repeated cross-validation.
<code>rep.times</code>	Number of repeated times for repeated cross-validation.
<code>pred.at</code>	Time point at which calibration should take place.
<code>ngroup</code>	Number of groups to be formed for calibration.
<code>seed</code>	A random seed for resampling.
<code>trace</code>	Logical. Output the calibration progress or not. Default is TRUE.

Examples

```
library("survival")

# Load imputed SMART data
data("smart")
x = as.matrix(smart[, -c(1, 2)])
time = smart$TEVENT
event = smart$EVENT
y = Surv(time, event)

# Fit Cox model with lasso penalty
lassofit = hdcox.lasso(x, y, nfolds = 5, rule = "lambda.1se", seed = 11)

# Model calibration by fitting the original data directly
cal.fitting = hdnom.calibrate(x, time, event, model.type = "lasso",
                             alpha = 1, lambda = lassofit$lasso_best_lambda,
                             method = "fitting",
                             pred.at = 365 * 9, ngroup = 5,
                             seed = 1010)

# Model calibration by bootstrap
# Normally boot.times should be set to 200 or more,
# we set it to 3 here only to save example running time.
cal.boot = hdnom.calibrate(x, time, event, model.type = "lasso",
                           alpha = 1, lambda = lassofit$lasso_best_lambda,
                           method = "bootstrap", boot.times = 3,
                           pred.at = 365 * 9, ngroup = 5,
                           seed = 1010)

# Model calibration by 5-fold cross-validation
cal.cv = hdnom.calibrate(x, time, event, model.type = "lasso",
                         alpha = 1, lambda = lassofit$lasso_best_lambda,
                         method = "cv", nfolds = 5,
                         pred.at = 365 * 9, ngroup = 5,
                         seed = 1010)

# Model calibration by repeated cross-validation
cal.repcv = hdnom.calibrate(x, time, event, model.type = "lasso",
                             alpha = 1, lambda = lassofit$lasso_best_lambda,
                             method = "repeated.cv", nfolds = 3, rep.times = 3,
                             pred.at = 365 * 9, ngroup = 5,
                             seed = 1010)

print(cal.fitting)
summary(cal.fitting)
plot(cal.fitting)

print(cal.boot)
summary(cal.boot)
plot(cal.boot)

print(cal.cv)
```

```
summary(cal.cv)
plot(cal.cv)

print(cal.repcv)
summary(cal.repcv)
plot(cal.repcv)
```

```
hdnom.compare.calibrate
```

Compare High-Dimensional Cox Models by Model Calibration

Description

Compare High-Dimensional Cox Models by Model Calibration

Usage

```
hdnom.compare.calibrate(x, time, event, model.type = c("lasso", "alasso",
  "flasso", "enet", "aenet", "mcp", "mnet", "scad", "snet"),
  method = c("fitting", "bootstrap", "cv", "repeated.cv"),
  boot.times = NULL, nfolds = NULL, rep.times = NULL, pred.at,
  ngroup = 5, seed = 1001, trace = TRUE)
```

Arguments

<code>x</code>	Matrix of training data used for fitting the model; on which to run the calibration.
<code>time</code>	Survival time. Must be of the same length with the number of rows as <code>x</code> .
<code>event</code>	Status indicator, normally 0 = alive, 1 = dead. Must be of the same length with the number of rows as <code>x</code> .
<code>model.type</code>	Model types to compare. Could be at least two of "lasso", "alasso", "flasso", "enet", "aenet", "mcp", "mnet", "scad", or "snet".
<code>method</code>	Calibration method. Could be "bootstrap", "cv", or "repeated.cv".
<code>boot.times</code>	Number of repetitions for bootstrap.
<code>nfolds</code>	Number of folds for cross-validation and repeated cross-validation.
<code>rep.times</code>	Number of repeated times for repeated cross-validation.
<code>pred.at</code>	Time point at which calibration should take place.
<code>ngroup</code>	Number of groups to be formed for calibration.
<code>seed</code>	A random seed for cross-validation fold division.
<code>trace</code>	Logical. Output the calibration progress or not. Default is TRUE.

Examples

```
# Load imputed SMART data
data(smart)
x = as.matrix(smart[, -c(1, 2)])
time = smart$TEVENT
event = smart$EVENT

# Compare lasso and adaptive lasso by 5-fold cross-validation
cmp.cal.cv =
  hdnom.compare.calibrate(x, time, event,
    model.type = c("lasso", "alasso"),
    method = "fitting",
    pred.at = 365 * 9, ngroup = 5, seed = 1001)

print(cmp.cal.cv)
summary(cmp.cal.cv)
plot(cmp.cal.cv)
```

```
hdnom.compare.validate
```

Compare High-Dimensional Cox Models by Model Validation

Description

Compare High-Dimensional Cox Models by Model Validation

Usage

```
hdnom.compare.validate(x, time, event, model.type = c("lasso", "alasso",
  "flasso", "enet", "aenet", "mcp", "mnet", "scad", "snet"),
  method = c("bootstrap", "cv", "repeated.cv"), boot.times = NULL,
  nfolds = NULL, rep.times = NULL, tauc.type = c("CD", "SZ", "UNO"),
  tauc.time, seed = 1001, trace = TRUE)
```

Arguments

<code>x</code>	Matrix of training data used for fitting the model; on which to run the validation.
<code>time</code>	Survival time. Must be of the same length with the number of rows as <code>x</code> .
<code>event</code>	Status indicator, normally 0 = alive, 1 = dead. Must be of the same length with the number of rows as <code>x</code> .
<code>model.type</code>	Model types to compare. Could be at least two of "lasso", "alasso", "flasso", "enet", "aenet", "mcp", "mnet", "scad", or "snet".
<code>method</code>	Validation method. Could be "bootstrap", "cv", or "repeated.cv".
<code>boot.times</code>	Number of repetitions for bootstrap.
<code>nfolds</code>	Number of folds for cross-validation and repeated cross-validation.

<code>rep.times</code>	Number of repeated times for repeated cross-validation.
<code>tauc.type</code>	Type of time-dependent AUC. Including "CD" proposed by Chambless and Diao (2006)., "SZ" proposed by Song and Zhou (2008)., "UNO" proposed by Uno et al. (2007).
<code>tauc.time</code>	Numeric vector. Time points at which to evaluate the time-dependent AUC.
<code>seed</code>	A random seed for cross-validation fold division.
<code>trace</code>	Logical. Output the validation progress or not. Default is TRUE.

References

- Chambless, L. E. and G. Diao (2006). Estimation of time-dependent area under the ROC curve for long-term risk prediction. *Statistics in Medicine* 25, 3474–3486.
- Song, X. and X.-H. Zhou (2008). A semiparametric approach for the covariate specific ROC curve with survival outcome. *Statistica Sinica* 18, 947–965.
- Uno, H., T. Cai, L. Tian, and L. J. Wei (2007). Evaluating prediction rules for t-year survivors with censored regression models. *Journal of the American Statistical Association* 102, 527–537.

Examples

```
# Load imputed SMART data
data(smart)
x = as.matrix(smart[, -c(1, 2)])[1:1000, ]
time = smart$TEVENT[1:1000]
event = smart$EVENT[1:1000]

# Compare lasso and adaptive lasso by 5-fold cross-validation
cmp.val.cv =
  hdnom.compare.validate(x, time, event,
                        model.type = c("lasso", "alasso"),
                        method = "cv", nfolds = 5, tauc.type = "UNO",
                        tauc.time = seq(0.25, 2, 0.25) * 365, seed = 1001)

print(cmp.val.cv)
summary(cmp.val.cv)
plot(cmp.val.cv)
plot(cmp.val.cv, interval = TRUE)
```

hdnom.external.calibrate

Externally Calibrate High-Dimensional Cox Models

Description

Externally Calibrate High-Dimensional Cox Models

Usage

```
hdnom.external.calibrate(object, x, time, event, x_new, time_new, event_new,
  pred.at, ngroup = 5)
```

Arguments

<code>object</code>	Model object fitted by <code>hdcox.*()</code> functions.
<code>x</code>	Matrix of training data used for fitting the model.
<code>time</code>	Survival time of the training data. Must be of the same length with the number of rows as <code>x</code> .
<code>event</code>	Status indicator of the training data, normally 0 = alive, 1 = dead. Must be of the same length with the number of rows as <code>x</code> .
<code>x_new</code>	Matrix of predictors for the external validation data.
<code>time_new</code>	Survival time of the external validation data. Must be of the same length with the number of rows as <code>x_new</code> .
<code>event_new</code>	Status indicator of the external validation data, normally 0 = alive, 1 = dead. Must be of the same length with the number of rows as <code>x_new</code> .
<code>pred.at</code>	Time point at which external calibration should take place.
<code>ngroup</code>	Number of groups to be formed for external calibration.

Examples

```
library("survival")

# Load imputed SMART data
data(smart)
# Use the first 1000 samples as training data
# (the data used for internal validation)
x = as.matrix(smart[, -c(1, 2)])[1:1000, ]
time = smart$TEVENT[1:1000]
event = smart$EVENT[1:1000]

# Take the next 1000 samples as external calibration data
# In practice, usually use data collected in other studies
x_new = as.matrix(smart[, -c(1, 2))][1001:2000, ]
time_new = smart$TEVENT[1001:2000]
event_new = smart$EVENT[1001:2000]

# Fit Cox model by lasso penalization
lassofit = hdcox.lasso(x, Surv(time, event), nfolds = 5, rule = "lambda.1se", seed = 11)

# External calibration
cal.ext =
  hdnom.external.calibrate(lassofit, x, time, event,
    x_new, time_new, event_new,
    pred.at = 365 * 5, ngroup = 5)

print(cal.ext)
```

```
summary(cal.ext)
plot(cal.ext, xlim = c(0.6, 1), ylim = c(0.6, 1))
```

```
hdnom.external.validate
```

Externally Validate High-Dimensional Cox Models with Time-Dependent AUC

Description

Externally Validate High-Dimensional Cox Models with Time-Dependent AUC

Usage

```
hdnom.external.validate(object, x, time, event, x_new, time_new, event_new,
  tauc.type = c("CD", "SZ", "UNO"), tauc.time)
```

Arguments

<code>object</code>	Model object fitted by <code>hdcox.*()</code> functions.
<code>x</code>	Matrix of training data used for fitting the model.
<code>time</code>	Survival time of the training data. Must be of the same length with the number of rows as <code>x</code> .
<code>event</code>	Status indicator of the training data, normally 0 = alive, 1 = dead. Must be of the same length with the number of rows as <code>x</code> .
<code>x_new</code>	Matrix of predictors for the external validation data.
<code>time_new</code>	Survival time of the external validation data. Must be of the same length with the number of rows as <code>x_new</code> .
<code>event_new</code>	Status indicator of the external validation data, normally 0 = alive, 1 = dead. Must be of the same length with the number of rows as <code>x_new</code> .
<code>tauc.type</code>	Type of time-dependent AUC. Including "CD" proposed by Chambless and Diao (2006)., "SZ" proposed by Song and Zhou (2008)., "UNO" proposed by Uno et al. (2007).
<code>tauc.time</code>	Numeric vector. Time points at which to evaluate the time-dependent AUC.

References

- Chambless, L. E. and G. Diao (2006). Estimation of time-dependent area under the ROC curve for long-term risk prediction. *Statistics in Medicine* 25, 3474–3486.
- Song, X. and X.-H. Zhou (2008). A semiparametric approach for the covariate specific ROC curve with survival outcome. *Statistica Sinica* 18, 947–965.
- Uno, H., T. Cai, L. Tian, and L. J. Wei (2007). Evaluating prediction rules for t-year survivors with censored regression models. *Journal of the American Statistical Association* 102, 527–537.

Examples

```

library("survival")

# Load imputed SMART data
data(smart)
# Use the first 1000 samples as training data
# (the data used for internal validation)
x = as.matrix(smart[, -c(1, 2)])[1:1000, ]
time = smart$TEVENT[1:1000]
event = smart$EVENT[1:1000]

# Take the next 1000 samples as external validation data
# In practice, usually use data collected in other studies
x_new = as.matrix(smart[, -c(1, 2)])[1001:2000, ]
time_new = smart$TEVENT[1001:2000]
event_new = smart$EVENT[1001:2000]

# Fit Cox model by lasso penalization
lassofit = hdcx.lasso(x, Surv(time, event), nfolds = 5, rule = "lambda.1se", seed = 11)

# External validation with time-dependent AUC
val.ext =
  hdnom.external.validate(lassofit, x, time, event,
                          x_new, time_new, event_new,
                          tauc.type = "UNO",
                          tauc.time = seq(0.25, 2, 0.25) * 365)

print(val.ext)
summary(val.ext)
plot(val.ext)

```

hdnom.kmplot

Kaplan-Meier Plot with Number at Risk Table for Internal Calibration and External Calibration Results

Description

Kaplan-Meier Plot with Number at Risk Table for Internal Calibration and External Calibration Results

Usage

```
hdnom.kmplot(object, group.name = NULL, time.at = NULL, col.pal = c("JCO",
  "Lancet", "NPG", "AAAS"))
```

Arguments

object An object returned by `hdnom.calibrate` or `hdnom.external.calibrate`.

group.name Risk group labels. Default is Group 1, Group 2, ..., Group k.
time.at Time points to evaluate the number at risk.
col.pal Color palette to use. Possible values are "JCO", "Lancet", "NPG", and "AAAS". Default is "JCO".

Examples

```
library("survival")

# Load imputed SMART data
data("smart")

# Use the first 1000 samples as training data
# (the data used for internal validation)
x = as.matrix(smart[, -c(1, 2)])[1:1000, ]
time = smart$TEVENT[1:1000]
event = smart$EVENT[1:1000]

# Take the next 1000 samples as external calibration data
# In practice, usually use data collected in other studies
x_new = as.matrix(smart[, -c(1, 2)])[1001:2000, ]
time_new = smart$TEVENT[1001:2000]
event_new = smart$EVENT[1001:2000]

# Fit Cox model by lasso penalization
lassofit = hdcx.lasso(x, Surv(time, event), nfolds = 5, rule = "lambda.1se", seed = 11)

### Internal calibration
cal.int = hdnom.calibrate(x, time, event, model.type = "lasso",
                        alpha = 1, lambda = lassofit$'lasso_best_lambda',
                        method = "cv", nfolds = 5,
                        pred.at = 365 * 9, ngroup = 3)

hdnom.kmplot(cal.int, group.name = c('High risk', 'Medium risk', 'Low risk'),
            time.at = 1:6 * 365)

### External calibration
cal.ext =
  hdnom.external.calibrate(lassofit, x, time, event,
                          x_new, time_new, event_new,
                          pred.at = 365 * 5, ngroup = 3)

hdnom.kmplot(cal.ext, group.name = c('High risk', 'Medium risk', 'Low risk'),
            time.at = 1:6 * 365)
```

Description

Log-Rank Test for Internal Calibration and External Calibration Results

Usage

```
hdnom.logrank(object)
```

Arguments

`object` An object returned by `hdnom.calibrate` or `hdnom.external.calibrate`.

Examples

```
library("survival")

# Load imputed SMART data
data("smart")

# Use the first 1000 samples as training data
# (the data used for internal validation)
x = as.matrix(smart[, -c(1, 2)])[1:1000, ]
time = smart$TEVENT[1:1000]
event = smart$EVENT[1:1000]

# Take the next 1000 samples as external calibration data
# In practice, usually use data collected in other studies
x_new = as.matrix(smart[, -c(1, 2)])[1001:2000, ]
time_new = smart$TEVENT[1001:2000]
event_new = smart$EVENT[1001:2000]

# Fit Cox model by lasso penalization
lassofit = hdcx.lasso(x, Surv(time, event), nfolds = 5, rule = "lambda.1se", seed = 11)

### Internal calibration
cal.int = hdnom.calibrate(x, time, event, model.type = "lasso",
                        alpha = 1, lambda = lassofit$'lasso_best_lambda',
                        method = "cv", nfolds = 5,
                        pred.at = 365 * 9, ngroup = 3)

hdnom.logrank(cal.int)

### External calibration
cal.ext =
  hdnom.external.calibrate(lassofit, x, time, event,
                          x_new, time_new, event_new,
                          pred.at = 365 * 5, ngroup = 3)

hdnom.logrank(cal.ext)
```

hdnom.nomogram *Nomograms for High-Dimensional Cox Models*

Description

Nomograms for High-Dimensional Cox Models

Usage

```
hdnom.nomogram(object, model.type = c("lasso", "alasso", "flasso", "enet",
  "aenet", "mcp", "mnet", "scad", "snet"), x, time, event, ddist,
  lambda = NULL, pred.at = NULL, fun.at = NULL, funlabel = NULL)
```

Arguments

<code>object</code>	Fitted model object.
<code>model.type</code>	Fitted model type. Could be one of "lasso", "alasso", "flasso", "enet", "aenet", "mcp", "mnet", "scad", or "snet".
<code>x</code>	Matrix of training data used for fitting the model.
<code>time</code>	Survival time. Must be of the same length with the number of rows as <code>x</code> .
<code>event</code>	Status indicator, normally 0 = alive, 1 = dead. Must be of the same length with the number of rows as <code>x</code> .
<code>ddist</code>	Data frame version of <code>x</code> , made by <code>datadist</code> .
<code>lambda</code>	Value of the penalty parameter <code>lambda</code> in <code>glmnet</code> or <code>ncvsurv</code> . Required except when <code>model.type == "flasso"</code> . We will use the selected variables at the provided <code>lambda</code> to build the nomogram, and make predictions. See the example for choosing a proper <code>lambda</code> value extracted from cross-validation results.
<code>pred.at</code>	Time point at which to plot nomogram prediction axis.
<code>fun.at</code>	Function values to label on axis.
<code>funlabel</code>	Label for fun axis.

Examples

```
library("rms")

# Load imputed SMART data
data(smart)
x = as.matrix(smart[, -c(1, 2)])
time = smart$TEVENT
event = smart$Eevent
y = Surv(time, event)

# Fit penalized Cox model with lasso penalty
lassofit = hdcx.lasso(x, y, nfolds = 5, rule = "lambda.1se", seed = 11)
```



```

# Prepare data needed for plotting nomogram
x.df = as.data.frame(x)
dd = datadist(x.df)
options(datadist = "dd")

# Generate hdnom.nomogram objects and plot nomogram
nom = hdnom.nomogram(lassofit$lasso_model, model.type = "lasso",
                    x, time, event, x.df,
                    lambda = lassofit$lasso_best_lambda, pred.at = 365 * 2,
                    funlabel = "2-Year Overall Survival Probability")

print(nom)
plot(nom)

```

hdnom.validate	<i>Validate High-Dimensional Cox Models with Time-Dependent AUC</i>
----------------	---

Description

Validate High-Dimensional Cox Models with Time-Dependent AUC

Usage

```

hdnom.validate(x, time, event, model.type = c("lasso", "alasso", "flasso",
      "enet", "aenet", "mcp", "mnet", "scad", "snet"), alpha, lambda,
      pen.factor = NULL, gamma, method = c("bootstrap", "cv", "repeated.cv"),
      boot.times = NULL, nfolds = NULL, rep.times = NULL,
      tauc.type = c("CD", "SZ", "UNO"), tauc.time, seed = 1001, trace = TRUE)

```

Arguments

<code>x</code>	Matrix of training data used for fitting the model; on which to run the validation.
<code>time</code>	Survival time. Must be of the same length with the number of rows as <code>x</code> .
<code>event</code>	Status indicator, normally 0 = alive, 1 = dead. Must be of the same length with the number of rows as <code>x</code> .
<code>model.type</code>	Model type to validate. Could be one of "lasso", "alasso", "flasso", "enet", "aenet", "mcp", "mnet", "scad", or "snet".
<code>alpha</code>	Value of the elastic-net mixing parameter <code>alpha</code> for <code>enet</code> , <code>aenet</code> , <code>mnet</code> , and <code>snet</code> models. For <code>lasso</code> , <code>alasso</code> , <code>mcp</code> , and <code>scad</code> models, please set <code>alpha = 1</code> . <code>alpha=1</code> : lasso (l1) penalty; <code>alpha=0</code> : ridge (l2) penalty. Note that for <code>mnet</code> and <code>snet</code> models, <code>alpha</code> can be set to very close to 0 but not 0 exactly.
<code>lambda</code>	Value of the penalty parameter <code>lambda</code> to use in the model fits on the resampled data. From the built Cox model.

<code>pen.factor</code>	Penalty factors to apply to each coefficient. From the built <i>adaptive lasso</i> or <i>adaptive elastic-net</i> model.
<code>gamma</code>	Value of the model parameter gamma for MCP/SCAD/Mnet/Snet models.
<code>method</code>	Validation method. Could be "bootstrap", "cv", or "repeated.cv".
<code>boot.times</code>	Number of repetitions for bootstrap.
<code>nfolds</code>	Number of folds for cross-validation and repeated cross-validation.
<code>rep.times</code>	Number of repeated times for repeated cross-validation.
<code>tauc.type</code>	Type of time-dependent AUC. Including "CD" proposed by Chambless and Diao (2006)., "SZ" proposed by Song and Zhou (2008)., "UNO" proposed by Uno et al. (2007).
<code>tauc.time</code>	Numeric vector. Time points at which to evaluate the time-dependent AUC.
<code>seed</code>	A random seed for resampling.
<code>trace</code>	Logical. Output the validation progress or not. Default is TRUE.

References

- Chambless, L. E. and G. Diao (2006). Estimation of time-dependent area under the ROC curve for long-term risk prediction. *Statistics in Medicine* 25, 3474–3486.
- Song, X. and X.-H. Zhou (2008). A semiparametric approach for the covariate specific ROC curve with survival outcome. *Statistica Sinica* 18, 947–965.
- Uno, H., T. Cai, L. Tian, and L. J. Wei (2007). Evaluating prediction rules for t-year survivors with censored regression models. *Journal of the American Statistical Association* 102, 527–537.

Examples

```
library("survival")

# Load imputed SMART data
data(smart)
x = as.matrix(smart[, -c(1, 2)])[1:500, ]
time = smart$TEVENT[1:500]
event = smart$EVENT[1:500]
y = Surv(time, event)

# Fit penalized Cox model with lasso penalty
lassofit = hdc Cox.lasso(x, y, nfolds = 5, rule = "lambda.1se", seed = 11)

# Model validation by bootstrap with time-dependent AUC
# Normally boot.times should be set to 200 or more,
# we set it to 3 here only to save example running time.
val.boot = hdnom.validate(x, time, event, model.type = "lasso",
                          alpha = 1, lambda = lassofit$lasso_best_lambda,
                          method = "bootstrap", boot.times = 3,
                          tauc.type = "UNO", tauc.time = seq(0.25, 2, 0.25) * 365,
```

```

seed = 1010)

# Model validation by 5-fold cross-validation with time-dependent AUC
val.cv = hdnom.validate(x, time, event, model.type = "lasso",
  alpha = 1, lambda = lassofit$lasso_best_lambda,
  method = "cv", nfolds = 5,
  tauc.type = "UNO", tauc.time = seq(0.25, 2, 0.25) * 365,
  seed = 1010)

# Model validation by repeated cross-validation with time-dependent AUC
val.repcv = hdnom.validate(x, time, event, model.type = "lasso",
  alpha = 1, lambda = lassofit$lasso_best_lambda,
  method = "repeated.cv", nfolds = 5, rep.times = 3,
  tauc.type = "UNO", tauc.time = seq(0.25, 2, 0.25) * 365,
  seed = 1010)

# bootstrap-based discrimination curves has a very narrow band
print(val.boot)
summary(val.boot)
plot(val.boot)

# k-fold cv provides a more strict evaluation than bootstrap
print(val.cv)
summary(val.cv)
plot(val.cv)

# repeated cv provides similar results as k-fold cv
# but more robust than k-fold cv
print(val.repcv)
summary(val.repcv)
plot(val.repcv)

```

hdnom.varinfo	<i>Extract Information of Selected Variables from High-Dimensional Cox Models</i>
---------------	---

Description

Extract the names and type of selected variables from established high-dimensional Cox models.

Usage

```
hdnom.varinfo(object, x)
```

Arguments

object	Model object fitted by <code>hdcox.*()</code> functions.
x	Data matrix used to fit the model.

Value

A list containing the index, name, type and range of the selected variables.

Examples

```
library("survival")

# Load imputed SMART data
data("smart")
x = as.matrix(smart[, -c(1, 2)])
time = smart$TEVENT
event = smart$EVENT
y = Surv(time, event)

# Fit Cox model by lasso penalization
lassofit = hdcox.lasso(x, y, nfolds = 5, rule = "lambda.1se", seed = 11)
hdnom.varinfo(lassofit, x)
```

plot.hdnom.calibrate *Plot Calibration Results*

Description

Plot Calibration Results

Usage

```
## S3 method for class 'hdnom.calibrate'
plot(x, xlim = c(0, 1), ylim = c(0, 1),
     col.pal = c("JCO", "Lancet", "NPG", "AAAS"), ...)
```

Arguments

x	An object returned by <code>hdnom.calibrate</code> .
xlim	x axis limits of the plot.
ylim	y axis limits of the plot.
col.pal	Color palette to use. Possible values are "JCO", "Lancet", "NPG", and "AAAS". Default is "JCO".
...	Other parameters for <code>plot</code> .

Examples

```
NULL
```

`plot.hdnom.compare.calibrate`*Plot Model Comparison by Calibration Results*

Description

Plot Model Comparison by Calibration Results

Usage

```
## S3 method for class 'hdnom.compare.calibrate'  
plot(x, xlim = c(0, 1), ylim = c(0, 1),  
     col.pal = c("JCO", "Lancet", "NPG", "AAAS"), ...)
```

Arguments

<code>x</code>	An object returned by <code>hdnom.compare.calibrate</code> .
<code>xlim</code>	x axis limits of the plot.
<code>ylim</code>	y axis limits of the plot.
<code>col.pal</code>	Color palette to use. Possible values are "JCO", "Lancet", "NPG", and "AAAS". Default is "JCO".
<code>...</code>	Other parameters (not used).

Examples

```
NULL
```

`plot.hdnom.compare.validate`*Plot Model Comparison by Validation Results*

Description

Plot Model Comparison by Validation Results

Usage

```
## S3 method for class 'hdnom.compare.validate'  
plot(x, interval = FALSE,  
     col.pal = c("JCO", "Lancet", "NPG", "AAAS"), ylim = NULL, ...)
```

Arguments

<code>x</code>	An object returned by <code>hdnom.compare.validate</code> .
<code>interval</code>	Show maximum, minimum, 0.25, and 0.75 quantiles of time-dependent AUC as ribbons? Default is <code>FALSE</code> .
<code>col.pal</code>	Color palette to use. Possible values are "JCO", "Lancet", "NPG", and "AAAS". Default is "JCO".
<code>ylim</code>	Range of y coordinates. For example, <code>c(0.5, 1)</code> .
<code>...</code>	Other parameters (not used).

Examples

NULL

```
plot.hdnom.external.calibrate
```

Plot External Calibration Results

Description

Plot External Calibration Results

Usage

```
## S3 method for class 'hdnom.external.calibrate'
plot(x, xlim = c(0, 1), ylim = c(0, 1),
     col.pal = c("JCO", "Lancet", "NPG", "AAAS"), ...)
```

Arguments

<code>x</code>	An object returned by <code>hdnom.external.calibrate</code> .
<code>xlim</code>	x axis limits of the plot.
<code>ylim</code>	y axis limits of the plot.
<code>col.pal</code>	Color palette to use. Possible values are "JCO", "Lancet", "NPG", and "AAAS". Default is "JCO".
<code>...</code>	Other parameters for <code>plot</code> .

Examples

NULL

```
plot.hdnom.external.validate
```

Plot Time-Dependent Discrimination Curves for External Validation

Description

Plot Time-Dependent Discrimination Curves for External Validation

Usage

```
## S3 method for class 'hdnom.external.validate'
plot(x, col.pal = c("JCO", "Lancet", "NPG",
  "AAAS"), ylim = NULL, ...)
```

Arguments

x	An object returned by <code>hdnom.external.validate</code> .
col.pal	Color palette to use. Possible values are "JCO", "Lancet", "NPG", and "AAAS". Default is "JCO".
ylim	Range of y coordinates. For example, <code>c(0.5, 1)</code> .
...	Other parameters (not used).

Examples

```
NULL
```

```
plot.hdnom.nomogram Plot Nomogram Objects Generated by hdnom.nomogram
```

Description

Plot Nomogram Objects Generated by `hdnom.nomogram`

Usage

```
## S3 method for class 'hdnom.nomogram'
plot(x, ...)
```

Arguments

x	An object returned by <code>hdnom.nomogram</code> .
...	Other parameters for <code>nomogram</code> .

Examples

```
NULL
```

`plot.hdnom.validate` *Plot Optimism-Corrected Time-Dependent Discrimination Curves for Validation*

Description

Plot Optimism-Corrected Time-Dependent Discrimination Curves for Validation

Usage

```
## S3 method for class 'hdnom.validate'
plot(x, col.pal = c("JCO", "Lancet", "NPG", "AAAS"),
     ylim = NULL, ...)
```

Arguments

<code>x</code>	An object returned by <code>hdnom.validate</code> .
<code>col.pal</code>	Color palette to use. Possible values are "JCO", "Lancet", "NPG", and "AAAS". Default is "JCO".
<code>ylim</code>	Range of y coordinates. For example, <code>c(0.5, 1)</code> .
<code>...</code>	Other parameters (not used).

Examples

```
NULL
```

`predict.hdcox.model` *Make Predictions from High-Dimensional Cox Models*

Description

Predict overall survival probability at certain time points from established Cox models.

Usage

```
## S3 method for class 'hdcox.model'
predict(object, x, y, newx, pred.at, ...)
```

Arguments

<code>object</code>	Model object fitted by <code>hdcox.*()</code> functions.
<code>x</code>	Data matrix used to fit the model.
<code>y</code>	Response matrix made with <code>Surv</code> .
<code>newx</code>	Matrix (with named columns) of new values for <code>x</code> at which predictions are to be made.
<code>pred.at</code>	Time point at which prediction should take place.
<code>...</code>	Other parameters (not used).

Value

A `nrow(newx) x length(pred.at)` matrix containing overall survival probability.

Examples

```
library("survival")

# Load imputed SMART data
data("smart")
x = as.matrix(smart[, -c(1, 2)])
time = smart$TEVENT
event = smart$Eevent
y = Surv(time, event)

lassofit = hdcox.lasso(x, y, nfolds = 5, rule = "lambda.1se", seed = 11)
predict(lassofit, x, newx = x[101:105, ], pred.at = 1:10 * 365)
```

`print.hdcox.model` *Print High-Dimensional Cox Model Objects*

Description

Print information about high-dimensional Cox models.

Usage

```
## S3 method for class 'hdcox.model'
print(x, ...)
```

Arguments

`x` Model object fitted by `hdcox.*()` functions.
`...` Other parameters (not used).

Examples

```
library("survival")

# Load imputed SMART data
data("smart")
x = as.matrix(smart[, -c(1, 2)])
time = smart$TEVENT
event = smart$Eevent
y = Surv(time, event)

lassofit = hdcox.lasso(x, y, nfolds = 5, rule = "lambda.1se", seed = 11)
print(lassofit)
```

```
print.hdnom.calibrate
```

Print Calibration Results

Description

Print Calibration Results

Usage

```
## S3 method for class 'hdnom.calibrate'  
print(x, ...)
```

Arguments

`x` An object returned by `hdnom.calibrate`.
`...` Other parameters (not used).

Examples

```
NULL
```

```
print.hdnom.compare.calibrate
```

Print Model Comparison by Calibration Results

Description

Print Model Comparison by Calibration Results

Usage

```
## S3 method for class 'hdnom.compare.calibrate'  
print(x, ...)
```

Arguments

`x` An object returned by `hdnom.compare.calibrate`.
`...` Other parameters (not used).

Examples

```
NULL
```

```
print.hdnom.compare.validate
```

Print Model Comparison by Validation Results

Description

Print Model Comparison by Validation Results

Usage

```
## S3 method for class 'hdnom.compare.validate'  
print(x, ...)
```

Arguments

x An object returned by `hdnom.compare.validate`.
... Other parameters (not used).

Examples

```
NULL
```

```
print.hdnom.external.calibrate
```

Print External Calibration Results

Description

Print External Calibration Results

Usage

```
## S3 method for class 'hdnom.external.calibrate'  
print(x, ...)
```

Arguments

x An object returned by `hdnom.external.calibrate`.
... Other parameters (not used).

Examples

```
NULL
```

```
print.hdnom.external.validate
```

Print External Validation Results

Description

Print External Validation Results

Usage

```
## S3 method for class 'hdnom.external.validate'  
print(x, ...)
```

Arguments

x An object returned by [hdnom.external.validate](#).
... Other parameters (not used).

Examples

```
NULL
```

```
print.hdnom.nomogram Print Nomograms Objects Generated by hdnom.nomogram
```

Description

Print Nomograms Objects Generated by hdnom.nomogram

Usage

```
## S3 method for class 'hdnom.nomogram'  
print(x, ...)
```

Arguments

x An object returned by [hdnom.nomogram](#).
... Other parameters for [nomogram](#).

Examples

```
NULL
```

```
print.hdnom.validate Print Validation Results
```

Description

Print Validation Results

Usage

```
## S3 method for class 'hdnom.validate'  
print(x, ...)
```

Arguments

`x` An object returned by `hdnom.validate`.
`...` Other parameters (not used).

Examples

```
NULL
```

```
smart Imputed SMART Study Data
```

Description

Imputed SMART study data (no missing values).

Usage

```
data(smart)
```

Format

A numeric matrix with 3,873 samples, and 29 variables (27 predictors + survival time + event):

- Demographics
 - SEX - gender
 - AGE - age in years
- Classical risk factors
 - SMOKING - smoking (never, former, current)
 - PACKYRS - in years
 - ALCOHOL - alcohol use (never, former, current)
 - BMI - Body mass index, in kg/m^2

- DIABETES
- Blood pressure
 - SYSTH - Systolic, by hand, in mm Hg
 - SYSTBP - Systolic, automatic, in mm Hg
 - DIASTH - Diastolic, by hand, in mm Hg
 - DIASTBP - Diastolic, automatic, in mm Hg
- Lipid levels
 - CHOL - Total cholesterol, in mmol/L
 - HDL - High-density lipoprotein cholesterol, in mmol/L
 - LDL - Low-density lipoprotein cholesterol, in mmol/L
 - TRIG - Triglycerides, in mmol/L
- Previous symptomatic atherosclerosis
 - CEREBRAL - Cerebral
 - CARDIAC - Coronary
 - PERIPH - Peripheral
 - AAA - Abdominal aortic aneurysm
- Markers of atherosclerosis
 - HOMOC - Homocysteine, in $\mu\text{mol/L}$
 - GLUT - Glutamine, in $\mu\text{mol/L}$
 - CREAT - Creatinine clearance, in mL/min
 - ALBUMIN - Albumin (no, micro, macro)
 - IMT - Intima media thickness, in mm
 - STENOSIS - Carotid artery stenosis > 50%

References

Steyerberg, E. W. (2008). Clinical prediction models: a practical approach to development, validation, and updating. Springer Science & Business Media.

Examples

```
data(smart)
dim(smart)

# # Code for generating the smart dataset by imputing the smarto data
# library("mice")
# library("Hmisc")
# data(smarto)
# set.seed(1000)
# smarti = aregImpute(~ I(TEVENT) + EVENT + SEX + I(AGE) +
#   DIABETES + CEREBRAL + CARDIAC + AAA + PERIPH +
#   STENOSIS + SYSTBP + DIASTBP + SYSTH + DIASTH +
#   I(LENGTH) + I(WEIGHT) + I(BMI) +
#   I(CHOL) + I(HDL) + I(LDL) + I(TRIG) + I(HOMOC) + I(GLUT) + I(CREAT) + I(IMT) +
#   as.factor(ALBUMIN) + as.factor(SMOKING) + I(PACKYRS) + as.factor(ALCOHOL),
#   n.impute = 10, data = smarto)
```

```
# imputed = impute.transcan(smarti, imputation = 1, data = smarto,
#   list.out = TRUE, pr = FALSE, check = FALSE)
# smart = smarto
# smart[names(imputed)] = imputed
# str(smart)
```

smarto

Original SMART Study Data

Description

Original SMART study data (with missing values) from Steyerberg et, al. 2008.

Usage

```
data(smarto)
```

Format

A numeric matrix with 3,873 samples, and 29 variables (27 predictors + survival time + event):

- Demographics
 - SEX - gender
 - AGE - age in years
- Classical risk factors
 - SMOKING - smoking (never, former, current)
 - PACKYRS - in years
 - ALCOHOL - alcohol use (never, former, current)
 - BMI - Body mass index, in kg/m^2
 - DIABETES
- Blood pressure
 - SYSTH - Systolic, by hand, in mm Hg
 - SYSTBP - Systolic, automatic, in mm Hg
 - DIASTH - Diastolic, by hand, in mm Hg
 - DIASTBP - Diastolic, automatic, in mm Hg
- Lipid levels
 - CHOL - Total cholesterol, in mmol/L
 - HDL - High-density lipoprotein cholesterol, in mmol/L
 - LDL - Low-density lipoprotein cholesterol, in mmol/L
 - TRIG - Triglycerides, in mmol/L
- Previous symptomatic atherosclerosis
 - CEREBRAL - Cerebral

- CARDIAC - Coronary
- PERIPH - Peripheral
- AAA - Abdominal aortic aneurysm
- Markers of atherosclerosis
 - HOMOC - Homocysteine, in $\mu\text{mol/L}$
 - GLUT - Glutamine, in $\mu\text{mol/L}$
 - CREAT - Creatinine clearance, in mL/min
 - ALBUMIN - Albumin (no, micro, macro)
 - IMT - Intima media thickness, in mm
 - STENOSIS - Carotid artery stenosis > 50%

References

Steyerberg, E. W. (2008). Clinical prediction models: a practical approach to development, validation, and updating. Springer Science & Business Media.

Examples

```
data(smarto)
dim(smarto)
```

```
summary.hdnom.calibrate
```

Summary of Calibration Results

Description

Summary of Calibration Results

Usage

```
## S3 method for class 'hdnom.calibrate'
summary(object, ...)
```

Arguments

object	An object returned by <code>hdnom.calibrate</code> .
...	Other parameters (not used).

Examples

```
NULL
```

`summary.hdnom.compare.calibrate`*Summary of Model Comparison by Calibration Results*

Description

Summary of Model Comparison by Calibration Results

Usage

```
## S3 method for class 'hdnom.compare.calibrate'  
summary(object, ...)
```

Arguments

<code>object</code>	An object returned by <code>hdnom.compare.calibrate</code> .
<code>...</code>	Other parameters (not used).

Examples

```
NULL
```

`summary.hdnom.compare.validate`*Summary of Model Comparison by Validation Results*

Description

Summary of Model Comparison by Validation Results

Usage

```
## S3 method for class 'hdnom.compare.validate'  
summary(object, silent = FALSE, ...)
```

Arguments

<code>object</code>	An object <code>hdnom.compare.validate</code> .
<code>silent</code>	Print summary table header or not, default is FALSE.
<code>...</code>	Other parameters (not used).

Examples

```
NULL
```

summary.hdnom.external.calibrate

Summary of External Calibration Results

Description

Summary of External Calibration Results

Usage

```
## S3 method for class 'hdnom.external.calibrate'  
summary(object, ...)
```

Arguments

object	An object returned by hdnom.external.calibrate .
...	Other parameters (not used).

Examples

NULL

summary.hdnom.external.validate

Summary of External Validation Results

Description

Summary of External Validation Results

Usage

```
## S3 method for class 'hdnom.external.validate'  
summary(object, silent = FALSE, ...)
```

Arguments

object	An object hdnom.external.validate .
silent	Print summary table header or not, default is FALSE.
...	Other parameters (not used).

Examples

NULL

`summary.hdnom.validate`*Summary of Validation Results*

Description

Summary of Validation Results

Usage

```
## S3 method for class 'hdnom.validate'  
summary(object, silent = FALSE, ...)
```

Arguments

<code>object</code>	An object <code>hdnom.validate</code> .
<code>silent</code>	Print summary table header or not, default is <code>FALSE</code> .
<code>...</code>	Other parameters (not used).

Examples

NULL

Index

`cv.glmnet`, 3, 5, 6, 8
`cv.ncvsurv`, 9–11, 13

`datadist`, 24

`glmnet`, 24

`hdcox.aenet`, 3
`hdcox.alasso`, 4
`hdcox.enet`, 5
`hdcox.flasso`, 7
`hdcox.lasso`, 8
`hdcox.mcp`, 9
`hdcox.mnet`, 10
`hdcox.scad`, 11
`hdcox.snet`, 12
`hdnom-package`, 3
`hdnom.calibrate`, 14, 21, 23, 28, 34, 40
`hdnom.compare.calibrate`, 16, 29, 34, 41
`hdnom.compare.validate`, 17, 30, 35, 41
`hdnom.external.calibrate`, 18, 21, 23, 30, 35, 42
`hdnom.external.validate`, 20, 31, 36, 42
`hdnom.kmplot`, 21
`hdnom.logrank`, 22
`hdnom.nomogram`, 24, 31, 36
`hdnom.validate`, 25, 32, 37, 43
`hdnom.varinfo`, 27

`ncvsurv`, 24
`nomogram`, 31, 36

`plot.hdnom.calibrate`, 28
`plot.hdnom.compare.calibrate`, 29
`plot.hdnom.compare.validate`, 29
`plot.hdnom.external.calibrate`, 30
`plot.hdnom.external.validate`, 31
`plot.hdnom.nomogram`, 31
`plot.hdnom.validate`, 32
`predict.hdcox.model`, 32
`print.hdcox.model`, 33
`print.hdnom.calibrate`, 34
`print.hdnom.compare.calibrate`, 34
`print.hdnom.compare.validate`, 35
`print.hdnom.external.calibrate`, 35
`print.hdnom.external.validate`, 36
`print.hdnom.nomogram`, 36
`print.hdnom.validate`, 37

`smart`, 37
`smarto`, 39
`summary.hdnom.calibrate`, 40
`summary.hdnom.compare.calibrate`, 41
`summary.hdnom.compare.validate`, 41
`summary.hdnom.external.calibrate`, 42
`summary.hdnom.external.validate`, 42
`summary.hdnom.validate`, 43
`Surv`, 3, 5–11, 13, 32